

1 CLAIMS

- 2
- 3 *Sub*
- 4 *11*
- 5 1. A method comprising:
- 6 a. determining when to generate a dump file; and
- 7 b. generating a dump file by gathering at least:
- 8 i. thread information for at least one running thread,
- 9 ii. context information for the thread,
- 10 iii. callstack information for the thread,
- 11 iv. process information for the process in which the thread is
- 12 running, and
- 13 v. information identifying a reason for generating the dump file.
- 14
- 15 2. The method as recited in Claim 1, wherein generating the dump file
- 16 further includes storing the dump file to a storage medium.
- 17
- 18 3. The method as recited in Claim 1, wherein generating the dump file
- 19 further includes gathering processor information about at least one
- 20 processor.
- 21
- 22 4. The method as recited in Claim 1, wherein determining when to
- 23 generate the dump file further includes determining that an exception
- 24 has occurred.
- 25

- 1 5. The method as recited in Claim 4, wherein the dump file does not  
2 further include any significant portion of a dynamically allocated  
3 memory.
- 4
- 5
- 6 6. The method as recited in Claim 5 wherein the dump file does not  
7 include any portion of a global initialized or uninitialized memory.
- 8
- 9 7. The method as recited Claim 5 wherein the dump file does not include  
10 any portion of the executable instructions used by the processor to  
11 execute the program.
- 12
- 13 8. The method as recited in Claim 1, wherein the dump file is a kernel  
14 minidump file associated with an operating system and the at least one  
15 running thread is the single thread which encountered the exception.
- 16
- 17
- 18 9. The method as recited in Claim 1, wherein the callstack information is  
19 a kernel stack.
- 20
- 21 10. The method as recited in Claim 1, wherein the process information  
22 identifies the process that initiated the thread.
- 23
- 24 11. The method as recited in Claim 1, further comprising:
- 25

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25

- a. allocating a buffer space in memory during an initialization process;  
and
- b. reserving space on a storage medium drive suitable for writing the  
contents of the buffer.

12. The method as recited in Claim 11, wherein:

- a. generating the dump file further includes initially storing the thread  
information, the context information, the callstack information, the  
process information, and the information identifying the reason for  
generating the dump file to the buffer space, and then copying the  
dump file from the buffer space to the storage medium as a dump  
file; and
- b. upon system re-initialization, transferring the dump file from the  
storage medium to at least one external computer.

13. The method as recited in Claim 12, further comprising upon re-  
initialization, after having stored the dump file to the storage medium,  
accessing the dump file on the storage medium and using at least a  
portion of the dump file to further understand an exception that was at  
least one reason for generating the dump file.

14. The method as recited in Claim 1, wherein the dump file is a user  
minidump file associated with at least one non-operating system  
program.

2

3

4

5

6

7

9

Q

10

•

- iv. process information for the process in which the thread is running, and

v. information identifying a reason for generating the ~~dump~~ file.

21. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes storing the dump file to a storage medium.

22. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes gathering processor information about at least one processor.

23. The computer-readable medium as recited in Claim 20, wherein determining when to generate the dump file further includes determining that an exception has occurred.

24. The computer-readable medium as recited in Claim 23, wherein the dump file does not further include any significant portion of a dynamically allocated memory.

25. The computer-readable medium as recited in Claim 24 wherein the dump file does not include any portion of a global initialized or uninitialized memory.

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

- computer-readable medium as recited Claim 24 wherein the dump file does not include any portion of the executable instructions to be executed by the processor to execute the program.
- computer-readable medium as recited in Claim 20, wherein the dump file is a kernel minidump file associated with an operating system and the at least one running thread is the single thread that encountered the exception.
- computer-readable medium as recited in Claim 20, wherein the stack information is a kernel stack.
- computer-readable medium as recited in Claim 20, wherein the process information identifies the process that initiated the thread.
- computer-readable medium as recited in Claim 20, comprising computer-executable instructions for performing steps of allocating a buffer space in memory during an initialization process, reserving space on a storage medium drive suitable for writing the contents of the buffer.
- computer-readable medium as recited in Claim 30, wherein: generating the dump file further includes initially storing the exception information, the context information, the callstack information

process information, and the information identifying the reason for generating the dump file to the buffer space, and then copying the dump file from the buffer space to the storage medium as a dump file; and upon system re-initialization, transferring the dump file from the storage medium to at least one external different computer.

32. The computer-readable medium as recited in Claim 31, further comprising computer-executable instructions for performing steps of, upon re-initialization after having stored the dump file to the storage medium, accessing the dump file on the storage medium and using at least a portion of the dump file to further understand an exception that was at least one reason for generating the dump file.

33. The computer-readable medium as recited in Claim 20, wherein the dump file is a user minidump file associated with at least one non-operating system program.

34. The computer-readable medium as recited in Claim 20, wherein generating the dump file further includes gathering callstack information for all running threads.

35. The computer-readable medium as recited in Claim 20, wherein the callstack information is a user callstack.

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

- computer-readable medium as recited in Claim 20, wherein the dump file further includes gathering processor context information for all running threads.
- computer-readable medium as recited in Claim 20, wherein the dump file further includes gathering a listing of all loaded modules for the faulting application program.
- computer-readable medium as recited in Claim 20, wherein the dump file is a directory indexed file that uses relative virtual addresses.
- system comprising memory, a data storage drive configured to store dump files to at least one data storage medium, and a processor coupled to the memory and the data storage drive and configured to:
- define when to generate a dump file; and
  - generate a dump file by gathering at least:
    - thread information for at least one running thread,
    - processor context information for the thread,
    - callstack information for the thread,
    - process information for the process in which the thread is running, and
    - information identifying a reason for generating the dump file.

- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25

- ment as recited in Claim 39, wherein generating the dump file further includes storing the dump file to a storage medium.
- ment as recited in Claim 39, wherein generating the dump file further includes gathering processor information about at least one processor.
- ment as recited in Claim 39, wherein determining whether the dump file further includes determining that an exception occurred.
- ment as recited in Claim 43, wherein the dump file further includes any significant portion of a dynamically allocated memory.
- ment as recited in Claim 43 wherein the dump file does not include any portion of a global initialized or uninitialized memory.
- ment as recited Claim 43 wherein the dump file does not include any portion of the executable instructions used by the processor to execute the program.
- ment as recited in Claim 39, wherein the dump file is a dump file associated with an operating system and the

1 least one running thread is the single thread which encountered the  
2 exception.

3  
4 47. The arrangement as recited in Claim 39, wherein the callstack  
5 information is a kernel stack.

6  
7 48. The arrangement as recited in Claim 39, wherein the process  
8 information identifies the process that initiated the thread.

9  
10 49. The arrangement as recited in Claim 39, further comprising computer-  
11 executable instructions for performing steps of:  
12 allocating a buffer space in memory during an initialization process;  
13 and  
14 reserving space on a storage medium drive suitable for writing the  
15 contents of the buffer.

16  
17 50. The arrangement as recited in Claim 49, wherein:  
18 generating the dump file further includes initially storing the thread  
19 information, the context information, the callstack information, the  
20 process information, and the information identifying the reason for  
21 generating the dump file to the buffer space, and then copying the dump  
22 file from the buffer space to the storage medium as a dump file; and  
23 upon system re-initialization, transferring the dump file from the  
24 storage medium to at least one external computer.  
25

- 1 51. The arrangement as recited in Claim 50, further comprising computer-  
2 executable instructions for performing steps of, upon re-initialization  
3 after having stored the dump file to the storage medium, accessing the  
4 dump file on the storage medium and using at least a portion of the  
5 dump file to further understand an exception that was at least one  
6 reason for generating the dump file.  
7
- 8 52. The arrangement as recited in Claim 39, wherein the dump file is a user  
9 minidump file associated with at least one non-operating system  
10 program.  
11
- 12 53. The arrangement as recited in Claim 39, wherein generating the dump  
13 file further includes gathering callstack information for all running  
14 threads.  
15
- 16 54. The arrangement as recited in Claim 39, wherein the callstack  
17 information is a user callstack.  
18
- 19 55. The arrangement as recited in Claim 39, wherein generating the dump  
20 file further includes gathering processor context information for all  
21 running threads.  
22
- 23 56. The arrangement as recited in Claim 39, wherein generating the dump  
24 file further includes gathering a listing of all loaded modules for the  
25 faulting application program.

CONFIDENTIAL

1  
2 57. The arrangement as recited in Claim 39, wherein the dump file is a  
3 directory indexed file that uses relative virtual addresses (RVAs).

4  
5 58. A method for generating a minimal dump file, the method comprising:

- 6 a. detecting an exception; and  
7 b. outputting:  
8 i. information on a faulting thread and an associated process,  
9 and  
10 ii. a list of loaded modules.

58-60  
OK  
for  
7/14/45  
AS

11  
12 59. The method as recited in Claim 58, further comprising storing the  
13 minimal dump file to a storage medium.

14  
15 60. The method as recited in Claim 58, further comprising transporting the  
16 minimal dump file using a communication resource.

17  
18 61. A method of communicating between a client process and a server  
19 process in a distributed processing system, comprising:

- 20 a. issuing, by the client process, a write dump file call having a  
21 plurality of call parameters comprising a process handle, a  
22 process identifier, a handle to a file where dump file information  
23 is to be written, and a dump type identifier;  
24 b. receiving, by the server process, the write dump file call and  
25 parsing the call to retrieve the parameters; and

7/29/201  
JF  
61-66

1 c. issuing, by the server process, a write dump file call  
2 acknowledgment providing a true-false indication.

3  
4 62. The method as recited in Claim 61, wherein the plurality of call  
5 parameters further includes a pointer to a structure describing an  
6 exception in the client that caused the dump file to be generated.

7  
8 63. The method as recited in Claim 61, wherein the plurality of call  
9 parameters further includes a pointer to an array of user data entry  
10 structures.

11  
12 64. The method as recited in Claim 61, wherein the plurality of call  
13 parameters further includes a pointer to a dump file callback data  
14 pointer.

15  
16 65. A method of communicating between a client process and a server  
17 process in a distributed processing system, comprising:

- 18 a. issuing, by the client process, a read dump file call having a  
19 plurality of call parameters comprising a header of a dump file  
20 and a data type identifier of data to read from a dump file;  
21 b. receiving, by the server process, the read dump file call and  
22 parsing the call to retrieve the parameters; and  
23 c. issuing, by the server process, a read dump file call  
24 acknowledgment providing a true-false indication and a plurality  
25 of call return parameters comprising a pointer to a beginning of a

dump stream, and a stream size identifying the size of the dump stream.

66. The method as recited in Claim 65, wherein the plurality of call return parameters further includes a pointer to a dump file directory.

~~add P2~~